University of Saskatchewan
Department of Computer Science

Cmpt 340

Final Examination

December 15, 1997

Time: 3 hours
Total Marks: 92

Professor: A. J. Kusalik
Closed Book[†]

Name: ____

Student Number: _______________________

Directions:

Answer each of the following questions in the space provided in <u>this</u> exam booklet. If you must continue an answer (e.g. in the extra space on the last page, or on the back side of a page), make sure you <u>clearly</u> indicate that you have done so and <u>where</u> to find the continuation.

Ensure that all written answers are <u>legible</u>; no marks will be given for answers which cannot be deciphered. Where a discourse or discussion is called for, please be concise and precise. If you find it necessary to make any assumptions to answer a question, please state the assumption with your answer.

Marks for each major question are given at the beginning of the question. There are a total of 92 marks.

The examination begins on the next page. Good luck.

For marking use only:

A.	_____/13	E.	_____/5	I.	_____/10
B.	_____/6	F.	_____/6	J.	_____/6
C.	_____/9	G.	_____/18		
D.	_____/8	H.	_____/11		
				Total:	______/92

[†] Closed book, except for one optional 8.5x11 inch quick reference sheet ("cheat sheet") of the student's <u>own</u> compilation.

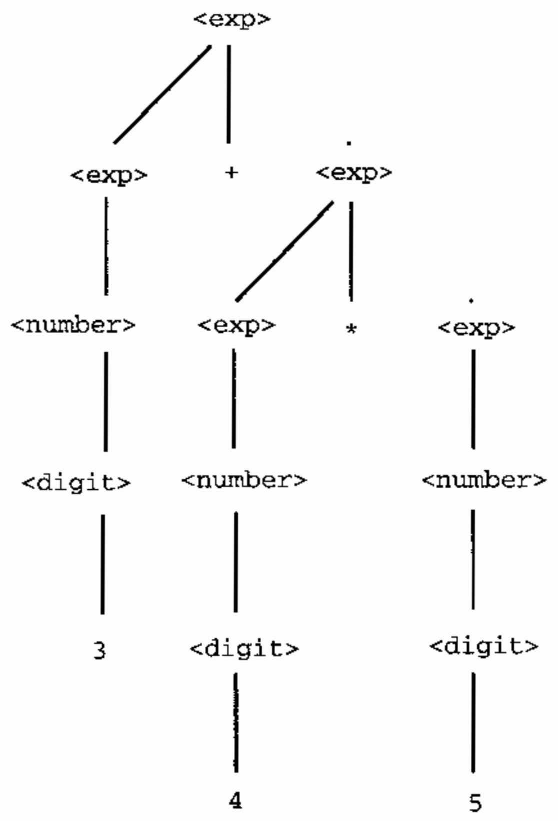
A. ***(1 marks each, 13 marks total)***
Indicate whether the following statements are **true** (T) or **false** (F).

___ Declarative languages are defined as those languages which have a formal, precise definition.

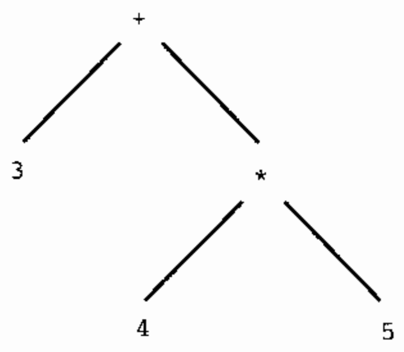
___ Suppose the expression

```
3 + 4 * 5
```

has a parse tree as follows:



This parse tree could be condensed to the following abstract syntax tree:



___ Simplicity (in programming language design) tends to run counter to generality, uniformity, orthogonality, and expressivity. That is because over-simplicity can make a language cumbersome to use, lacking in expressiveness, and subject to too many restrictions.

___ Context-free languages are not hampered by the context in which a particular syntactic construct appears. Therefore, context-free languages are more powerful than context-sensitive languages.

___ DCGs are transformed to Prolog clauses by adding 3 additional arguments. The first represents (holds) the parse tree, the second represents a "left label", and the third represents the "right label".

___ Global variables are an example of a type of variable whose storage binding is static.

___ A symbol table can be regarded abstractly as a (mathematical) function from (a set of) names to (a set of) attributes.

___ Inference rules in logic (as used for specifying axiomatic semantics) can be written

$$\frac{conclusion}{premise}$$

as in

$$\frac{a \rightarrow b,\ b \rightarrow c}{a \rightarrow c}$$

___ The situation of heap space running into the runtime stack space is indicated by the dynamic link and the static link in a new activation record being the same value.

___ "Programs" in logic programming languages like Prolog consist of sets of facts and rules, and follow-up queries or goals that are solved by an inference engine.

___ Applicative-order evaluation (in functional languages) corresponds to pass-by-value.

___ Referential transparency is a property of languages in which pointers to dynamically-allocated data structures are made transparent; i.e. the user does not need to be aware of the presence of the pointers and can manipulate the dynamically-allocated data structures without accessing or dealing with pointers.

___ There is no single best programming language.

B. ***(2+1+1+2 = 6 marks)***
Each of the following questions requires a very short, concise answer.

1. Prograph incorporates and exemplifies several programming language paradigms. However, two of the paradigms not well-illustrated by, or not present in, the other languages we have studied. What are those two paradigms?

2. In the programming language C, the operators `&&` and `||` are known as "short circuit" versions of the boolean *and* and *or* operators, respectively. That is, they evaluate their second argument only if it is necessary. For example, in the (model) statement

```
if ( D && E ) S
```

E is only evaluated if *D* is true. Otherwise (if *D* is false), there is no need to evaluate *E* since the result of *D*&&*E* will always be false.

These short-circuit operators are similar to what construct or feature in functional languages?

3. Abstract machines play a vital role in implementation of programming languages. Name one other aspect of programming languages in which abstract machines are used.

Note: answers related to "interpreters", "pseudo-interpreters", "runtime system", "operating system", etc. are all related to implementation. Name a use of abstract machines in a different area of programming language design and study.

4. Here are two opposing statements regarding programming languages:

(a) A program should never run with error checking turned off. Turning off error checking after a program has been tested is like throwing away the life preserver when graduating from the pool to the ocean.

(b) A program should always have error checking turned off after the testing stage. Keeping error checking on is like keeping training wheels on a bike after you've entered a race.

Specifically, what language criterion (or criteria) does (a) support? Which does (b) support?

C. ***(3 marks each, 9 marks total)***
Answer each of the following questions with a short, concise answer.

1. What does a linker do?

2. What is the difference between static semantics and dynamic semantics?

3. What is the difference between a syntactic domain and a semantic domain (in the context of denotational semantics)?

D. ***(4 marks each, 8 marks total)***
Answer each of the following questions with a concise answer.

1. Consider the following PASCAL program fragment:

```
procedure example( n : real );
   var answer : real;
   procedure util( r : real ) : real;
      var s, t : real;
      procedure aux( v : real ) : real;
         var temp : real;
      begin
          . . .
      end {aux};
   begin
       . . .
      s := aux( r );   (*1*)
       . . .
      t := util( s ); (*2*)
       . . .
   end {util};
begin
    . . .
   answer := util( n ); (*3*)
    . . .
end {example};
```

(a) If the static nesting level of statement labelled `(*1*)` is 3, what is the static nesting level associated with the procedure (name) aux?

(b) What is the static distance between the call of procedure `util` in the statement labelled `(*2*)` and the declaration of the procedure name `util` (definition of the procedure `util`)?

2. Name 4 fields that are often present in activation records for/of statically-scoped, procedural languages.

E. ***(5 marks)***

Consider the following subprogram in a statically-scoped, Pascal-like language:

```
real procedure p( k, first, last, ak );
  value first, last;      /* first and last are passed-by-value */
  name k, ak;             /* k and ak are passed-by-name */
  integer k, first, last;
  real ak;
begin
  real S;

  S := 0;
  for k := first step 1 until last do
     S := S + ak;
  p := S;            /* return value of S */
end;
```

As described by the program comments, variables `k` and `ak` are pass-by-name parameters, and `first` and `last` are passed by value. Such a construction is known as Jensen's device.

1. What does the call

```
x := p( i, 1, n, V[i] );
```

compute? Give your answer in a mathematically succinct form (as a mathematical expression).

2. What does the call

```
x := p( i, 1, m, p( j, 1, n, A[i,j] ) );
```

compute? Again, give your answer in a mathematically succinct form (as a mathematical expression).

F. ***(6 marks)***

Consider the following function `inc_n`. `inc_n` takes as argument an integer n and returns the n-th increment function. The latter function, when given an integer argument, increments that value (the argument) by n and returns the result. `inc_n` is defined as follows (in Gofer):

```
inc_n n = (+ n)
```

and here is an example of it being used:

```
(inc_n 3) 2
5 :: Int
```

1. What is the type of the expression `(inc_n 3)` ?

2. Express the function `(inc_n 3)` as a lambda expression

3. What is the type of the function `inc_n` (assuming the argument n will always be of type `Int`)?

G. ***(3 marks each, 18 marks total)***

For each of the following pairs of languages, state and describe one nontrivial feature or characteristic that they have in common. I.e. describe one way that the two languages (of each pair) are the same in a significant or distinctive way. Do not focus on a particular implementation, but on the language in general.

A good way to help make your point in each case would be to use program samples, or example statements or expressions.

1. Smalltalk and Prolog

2. Gofer and Prograph

3. Prograph and Smalltalk

4. Prolog and Prograph

5. Prolog and Gofer

6. A constraint logic programming language (such as CLP(R)) and Prolog

H. ***(4+4+3 = 11 marks)***
Answer each of the following questions with a discussion-oriented answer.

1. Recall the concept of a pseudo-interpreter (or hybrid implementation of a language) in which the compiler is written in the object language itself (i.e. if the implementation is for a language L, the compiler taking L to virtual machine code is also written in L). As discussed in class, one advantage of such a scheme is that it aids in portability.

   A Prolog system called BinProlog (another Canadian product, this time from the University of Moncton!) has an interesting implementation. Part of the implementation is a pseudo-interpreter: it includes a virtual machine, and a compiler — written in Prolog — which translates Prolog to virtual machine code. However, the implementation goes further:

   (i) the compiler (given Prolog code as input) can generate (can compile to) C source code, and

   (ii) the virtual machine is written in Prolog.

How do these two additional features ((i) and (ii) above) affect portability? E.g. is this Prolog implementation more portable or less portable than the usual implementation (described in the first paragraph)? Why?

2. In class presentation of material on activation records, activation records were taken to be of statically-determined, fixed size.

Consider a language implemented using activation records which has the following feature. The language supports fixed-length arrays where the size of the array can be determined as late as run-time entry of the procedure in which the array is defined. Algol-60, for example, allowed such dynamic-length arrays.

What effect would such a feature have on procedure call and return (using activation records)? What would a compiler do differently in the case of a procedure which used a dynamic-length array as described above?

3. Smalltalk's convention for message selectors of arithmetic operators (e.g. '+') violates the regularity principle. I.e. it violates the general rule for message selector format for the language. For example, if an object, say x, is to get a message `push:` with argument `i`, one writes

```
x push: i
```

Or if an object `circle` is to get a message `circleAt:withRadius:` with arguments `c` and `5`, one would write

```
 circle circleAt: c withRadius: 5
```

However, to send a message + with argument `3` to the (integer) object `4`, one writes

```
 4 + 3
```

Is this violation of the regularity principle a justifiable violation? Why or why not?

I. (5 marks each, 10 marks total)

Answer each of the following questions with a discussion-oriented answer.

1. Give a programming example of a side-effect, i.e. a program sample which demonstrates a side-effect. You can use C, Pascal, or one of the languages covered in class for specifying the sample. To properly satisfy the definition of a side-effect, your construction (your example) should be potentially harmful or dangerous. Make sure to explain your example, and how it is potentially harmful or dangerous.

2. Describe how branching is achieved in Prograph; i.e. how different code is executed upon different conditions holding (or not holding). You may use an example or small diagrams to augment your description.

J. ***(6 marks)***
Consider the four languages (or language types) studied in the last part of the class; e.g. Prolog and CLP, Gofer, Prograph, and Smalltalk. Consider the runtime systems necessary to support each of these languages. Give an ordering for the complexity of these runtime systems, and a justification for your ordering. I.e. order the languages from least complex to most complex in terms of the necessary sophistication and functionality of their runtime (support) systems, state that ordering, and explain why you chose it. Illustrate with examples if you wish.

Note: focus on the runtime support systems of the languages, and not on other things such as the language paradigm, virtual machine, or resource utilization.